**PRA SKRIPSI**

**PEMBUATAN *FRAMEWORK* *REMOTE PROCEDURE CALL* BERBASIS NODEJS UNTUK KOMUNIKASI DATA *CLIENT-SERVER* PADA APLIKASI MANAJEMEN KARYAWAN**

****

**HADI HIDAYAT HAMMURABI**

**NIM: 155410097**

**PROGRAM STUDI INFORMATIKA**

**PROGRAM SARJANA**

**FAKULTAS TEKNOLOGI INFORMASI**

**UNIVERSITAS TEKNOLOGI DIGITAL INDONESIA**

**YOGYAKARTA**

**2023**

**PRA SKRIPSI**

**PEMBUATAN *FRAMEWORK* *REMOTE PROCEDURE CALL* BERBASIS NODEJS UNTUK KOMUNIKASI DATA *CLIENT-SERVER* PADA APLIKASI MANAJEMEN KARYAWAN**

**Diajukan sebagai salah satu syarat untuk menyelesaikan studi**

**Program Sarjana**

**Program Studi Informatika**

**Fakultas Teknologi Informasi**

**Universitas Teknologi Digital Indonesia**

**Yogyakarta**

**Disusun Oleh**

**HADI HIDAYAT HAMMURABI**

**NIM: 155410097**

**PROGRAM STUDI INFORMATIKA**

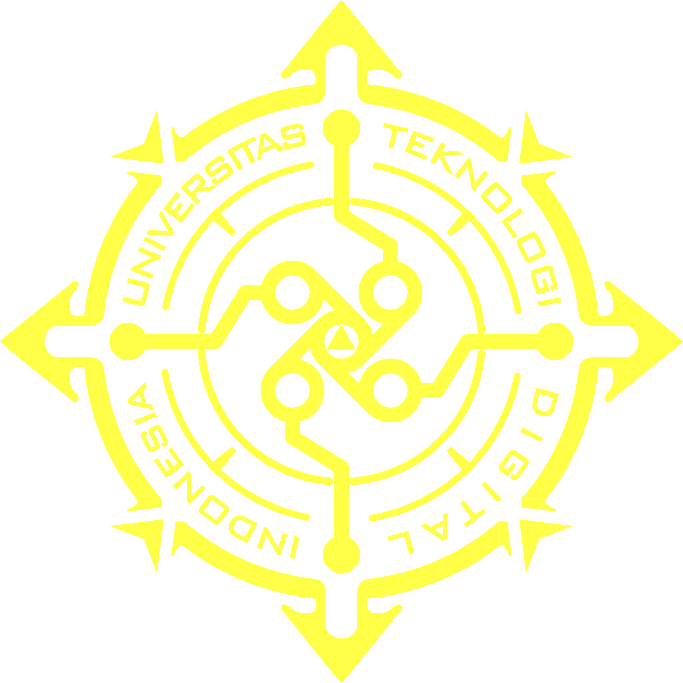
**PROGRAM SARJANA**

**FAKULTAS TEKNOLOGI INFORMASI**

**UNIVERSITAS TEKNOLOGI DIGITAL INDONESIA**

**YOGYAKARTA**

**2023**



# HALAMAN PERSETUJUAN

|  |  |  |
| --- | --- | --- |
| **Judul** | : | PEMBUATAN *FRAMEWORK* *REMOTE PROCEDURE CALL* BERBASIS NODEJS UNTUK KOMUNIKASI DATA *CLIENT-SERVER* PADA APLIKASI MANAJEMEN KARYAWAN |
| **Nama** | : | Hadi Hidayat Hammurabi |
| **NIM** | : | 155410097 |
| **Program Studi** | : | Informatika |
| **Program** | : | Sarjana |
| **Semester** | : | Gasal |
| **Tahun Akademik** | : | 2022/2023 |

Telah diperiksa dan disetujui untuk diseminarkan di hadapan dosen penguji seminar PRA SKRIPSI.

Yogyakarta, ………………

Dosen Pembimbing

Thomas Edyson Tarigan, S. Kom., M. Cs.

NIDN: 0023107402

# KATA PENGANTAR

Puji syukur penulis panjatkan ke hadirat Allah SWT yang telah memberi rahmat dan hidayahnya-Nya sehingga penyusunan laporan Pra Skripsi ini dapat diselesaikan.

Tentunya, penulisan laporan Pra Skripsi ini tidak lepas dari dukungan dan peran serta berbagai pihak. Atas adanya bantuan tersebut, penulis hendak menyampaikan terima kasih kepada beberapa pihak, di antaranya sebagai berikut:

1. Bapak Ir. Totok Suprawoto, M.M., M.T. sebagai Rektor Universitas Teknologi Digital Indonesia.
2. Bapak Thomas Edyson Tarigan, S. Kom., M. Cs. selaku dosen pembimbing Pra Skripsi.

Laporan ini penulis susun untuk sebagai salah satu syarat untuk menyelesaikan studi jenjang Strata 1 program studi Informatika serta untuk memperoleh gelar Sarjana Komputer di Universitas Teknologi Digital Indonesia.

Penulis menyadari bahwa dalam penulisan laporan Pra Skripsi ini tentu terdapat banyak kekurangan dan jauh dari kata sempurna. Oleh karena itu, penulis mengharapkan kritik dan saran yang bersifat konstruktif dengan harapan agar dapat menjadi lebih baik lagi. Semoga laporan Pra Skripsi ini memberikan manfaat bagi penulis dan tentu pula bagi pembaca.

Yogyakarta, ……………...

(Penulis)

# DAFTAR ISI

HALAMAN PERSETUJUAN ii

KATA PENGANTAR iii

DAFTAR ISI iv

PENDAHULUAN 6

1.1. Latar Belakang 6

1.2. Rumusan Masalah 7

1.3. Ruang Lingkup 7

1.4. Tujuan Penelitian 8

1.5. Manfaat Penelitian 8

TINJAUAN PUSTAKA DAN DASAR TEORI 9

2.1. Tinjauan Pustaka 9

2.2. Dasar Teori 11

2.2.1. Protokol HTTP 11

2.2.2. Komunikasi Data Client-Server 12

2.2.3. Remote Procedure Call (RPC) 12

2.2.4. Framework Aplikasi 13

2.2.5. Javascript 13

2.2.6. NodeJS 14

2.2.7. Javascript Object Notation (JSON) 15

METODE PENELITIAN 16

3.1. Peralatan 16

3.1.1. Perangkat Keras 16

3.1.2. Perangkat Lunak 16

3.2. Perancangan Framework 17

3.2.1. Komponen Utama Framework 17

3.2.2. RPC *Server* 18

3.2.3. RPC *Client* 19

3.3. Perancangan Aplikasi 20

3.3.1. Arsitektur Sistem 21

3.3.2. Diagram Usecase 22

3.3.3. Rancangan Tabel 23

3.3.4. Rancangan Antarmuka 24

IMPLEMENTASI DAN PEMBAHASAN 28

4.1. Implementasi *Framework* 28

4.1.1. *Igniter* di RPC *Server* 28

4.1.2. *Procedure* di RPC *Server* 29

4.1.3. *Server* di RPC *Server* 31

4.1.4. *Connector* di RPC *Client* 32

4.1.5. *Caller* di RPC *Client* 32

4.2. Implementasi Sistem 33

4.3. Tampilan Sistem 34

4.3.1. Halaman Depan 34

4.3.2. Halaman Tambah Data 34

4.3.3. Halaman Ubah Data 35

4.3.4. Halaman Hapus Data 35

5.1. Simpulan 36

5.2. Saran 36

DAFTAR PUSTAKA 37

BAB I

# PENDAHULUAN

## Latar Belakang

Komunikasi data *client-server* merupakan aktivitas berkirim data antara aplikasi di sisi *client* (pengguna) dengan aplikasi yang berjalan di sisi *server*. Hal ini memungkinkan dua atau lebih *client* memiliki suatu pusat data agar seluruh data dan informasi dalam suatu aplikasi dapat terintegrasi.

Di masa kini, perkembangan informasi sangat pesat yang menjadi penyebab munculnya berbagai metode untuk mengimplementasi komunikasi data *client-server* ini. Atas pengaruh tersebut, teknologi pemrograman juga ikut berkembang sehingga bermunculan bahasa pemrograman dan teknologi baru.

Kebutuhan terhadap aplikasi juga semakin banyak sehingga dibutuhkan sebuah *framework* untuk mempermudah dan mempercepat pengembangan aplikasi. Dengan adanya *framework*, para pengembang aplikasi dapat dipermudah karena adanya struktur yang rapi sehingga perawatan aplikasi menjadi mudah, modul-modul yang disediakan sangat membantu dalam menyelesaikan suatu masalah, dan banyak lagi keuntungan lainnya.

Salah satu metode komunikasi data yang dapat diterapkan untuk pengembangan aplikasi berbasis *client-server* adalah *Remote Procedure Call* yang umum disingkat RPC. Metode ini digunakan untuk memanggil suatu prosedur yang ada di komputer lain, dalam hal ini komputer *client* memanggil prosedur yang ada di komputer *server*. Implementasi metode ini dapat dilakukan di berbagai teknologi pemrograman, seperti Javascriptuntuk *client*, dengan bantuan NodeJS untuk bagian *server*. Dengan demikian, pemenuhan kebutuhan *framework* untuk mempermudah implementasi RPC ini lah yang mendorong penulis untuk meneliti “PEMBUATAN *FRAMEWORK* *REMOTE PROCEDURE CALL* BERBASIS NODEJS UNTUK KOMUNIKASI DATA *CLIENT-SERVER* PADA APLIKASI MANAJEMEN KARYAWAN”. Nantinya, *framework* ini diharapkan dapat memudahkan para pengembang aplikasi dalam mengembangkan aplikasi berbasis *client-server* dengan metode RPC.

## Rumusan Masalah

Berdasarkan uraian latar belakang tersebut, maka yang menjadi rumusan masalah yaitu bagaimana mengimplementasi RPC dalam suatu *framework* menggunakan NodeJS untuk studi kasus aplikasi manajemen karyawan.

## Ruang Lingkup

Adapun ruang lingkup permasalahan dalam penelitian ini, yaitu:

1. Membuat sebuah *framework* berbasis NodeJS.
2. *Framework* ini dibuat untuk implementasi metode RPC.
3. Protokol yang digunakan adalah HTTP.
4. *Framework* diterapkan pada aplikasi manajemen karyawan.
5. Manajemen yang dilakukan meliputi operasi pengambilan, penambahan, pengubahan, dan penghapusan data karyawan.
6. *Framework* akan diuji berdasarkan pengujian fungsionalitas.
7. Versi NodeJS yang digunakan v18.14.0.

## Tujuan Penelitian

Tujuan dari penelitian ini adalah untuk membangun sebuah *framework* RPC berbasis NodeJS sebagai media komunikasi antara *client* dengan *server*.

## Manfaat Penelitian

Manfaat dari penelitian ini, yaitu:

1. Mengetahui perancangan sebuah *framework* RPC untuk NodeJS.
2. Menghasilkan *framework* yang dapat membantu para pengembang dalam mengimplementasi metode RPC.

BAB II

# TINJAUAN PUSTAKA DAN DASAR TEORI

## 2.1. Tinjauan Pustaka

Berkembangnya teknologi informasi, terutama aplikasi web, menunjukkan kebutuhan teknologi semakin meluas. Hal ini tampak pada penelitian Wawan Setiawan tahun 2017, kebutuhan untuk absensi siswa dapat dilakukan dengan menggunakan teknologi berupa aplikasi web. Pada penelitian ini, penulis menggunakan teknologi NodeJS untuk membuat seluruh bagian dari aplikasi tersebut.

Berbagai usaha dilakukan agar dapat memenuhi kebutuhan lainnya, termasuk peningkatan arsitektur aplikasi dari segi struktur kode. Menurut Iqbal Sulistyo pada penelitiannya di tahun 2017, bahwa struktur kode *Model-View-Controller* (MVC) dapat membantu pengembang dalam membuat aplikasi dengan cepat dan mudah. Begitu pula pada penelitian Yosafat Aria Negara tahun 2018, arsitektur aplikasi dibuat dengan menerapkan konsep *client-server* untuk memisahkan kode pada sisi client dengan sisi server yang praktiknya digunakan untuk membangun aplikasi layanan kamar hotel.

Implementasi konsep *client-server* ini dapat dilakukan dengan beberapa cara, salah satunya adalah dengan menggunakan *web service*. Dengan demikian, layanan-layanan menjadi terbuka dan mudah diakses untuk kepentingan integrasi data dan kolaborasi informasi yang bisa diakses melalui internet, hal ini termuat dalam penelitian yang dilakukan oleh Lalu Himawan Satraji pada tahun 2017.

Kemudian, *web service* juga dapat dibuat dalam berbagai bentuk dan jenisnya, salah satunya adalah REST API. Seperti hasil penelitian Riko tahun 2019 yang mengimplementasi REST API untuk membuat aplikasi penjadwalan ujian pendadaran dan seminar proposal, dikatakan bahwa REST API digunakan untuk memfasilitasi pertukaran informasi atau data antara dua atau lebih aplikasi.

Pada penelitian ini, membuat *framework* *client-server* yang menerapkan metode *Remote Procedure Call* (RPC) dengan menggunakan bahasa pemrograman Javascript dan dijalankan dengan NodeJS sebagai servernya. Implementasi *client-server* di sini menerapkan protokol yang sama dengan *web service* dan salah satu metode pengiriman data yang terdapat pada REST API. Pembuatan *framework* ini bertujuan untuk mempermudah dalam melakukan pengembangan aplikasi berbasis *client-server*, terutama metode RPC.

Perbandingan dari beberapa penelitian di atas, sekaligus menjadi acuan dalam pengembangan *framework* ini ditampilkan pada Tabel 2.1.

Tabel 2.1. Tinjauan Pustaka

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Penulis** | **Teknologi/**  **Metode** | **Objek** | **Masalah** | **Hasil** |
| Setiawan Wawan  (2017) | NodeJS | Absensi Siswa | Absensi dilakukan manual oleh guru piket | Aplikasi web presensi online |
| Iqbal Sulistyo  (2017) | *Model-View-Controller* | Penanganan Tilang | Pelaksanaan tilang yang masih konvensional | Sistem informasi tilang |
| Lalu Himawan Satraji  (2017) | *Web Service* | Delivery Makanan Khas Yogyakarta | Pemesanan delivery untuk makanan khas Yogyakarta masih belum ada | Aplikasi delivery makanan khas Yogyakarta |
| Yosafat Aria Negara  (2018) | *Client-Server* | Layanan Kamar Hotel | Pelayanan hotel yang masih manual | Aplikasi layanan kamar hotel |
| Riko  (2019) | REST API | Sistem Penjadwalan Pendadaran dan Seminar Proposal | Penjadwalan ujian pendadaran dan seminar proposal yang masih dilakukan manual oleh akademik | Sistem informasi penjadwalan pendadaran dan seminar proposal |
| Hadi Hidayat Hammurabi  (2021) | NodeJS, RPC | Aplikasi Berbasis RPC | Pengembangan aplikasi RPC yang belum banyak didukung *framework* | *Framework* RPC NodeJS |

## 2.2. Dasar Teori

### 2.2.1. Protokol HTTP

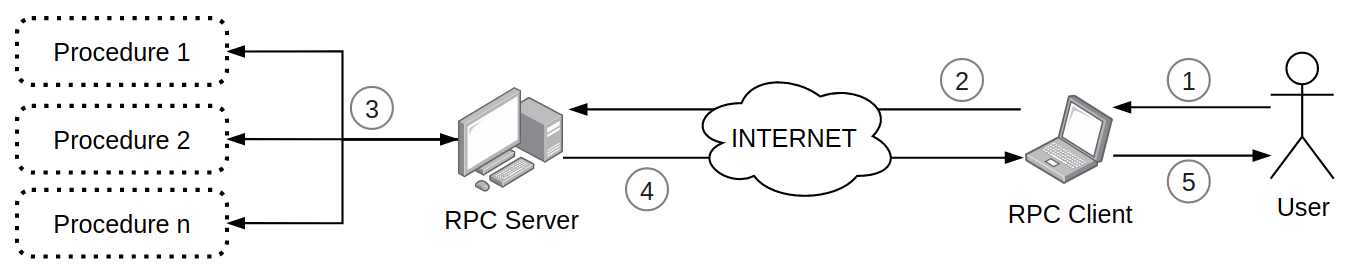
HTTP atau *Hyper Text Transfer Protocol* merupakan protokol yang sangat sederhana, berdasarkan skema yang sudah dikenal dari serangkaian kecil perintah yang dikeluarkan oleh *client* (*web browser*) dan informasi terkait dengan bentuk kode balasan yang dikeluarkan oleh *server* (situs *web*). (Goralski. W, 2017)

### 2.2.2. Komunikasi Data Client-Server

*Client* digambarkan sebagai program yang meminta untuk administrasi atau data, dan *server* digambarkan sebagai prosedur atau aplikasi yang memberikan administrasi atau data. Administrasi atau data yang diminta dan diberikan antara *server* dan *client* dapat menjadi aset, misalnya informasi, dokumen, objek, perangkat tampilan, atau kontrol. Contoh kerangka kerja *client-server* adalah program web, pengelolaan akun, dan kerangka email. (Shobhika Sejwal, 2019)

### 2.2.3. Remote Procedure Call (RPC)

RPC memungkinkan pengguna untuk melakukan panggilan ke prosedur jarak jauh yang berada di ruang proses lain. Proses ini dapat berjalan di mesin yang sama atau mesin lain di jaringan. Mekanisme RPC banyak digunakan dalam membangun sistem terdistribusi karena mereka mengurangi kompleksitas sistem dan biaya pengembangan. Tujuan utama RPC adalah membuat panggilan prosedur jarak jauh menjadi transparan bagi pengguna. Dengan kata lain, ini memungkinkan pengguna untuk melakukan panggilan prosedur jarak jauh seperti panggilan prosedur lokal. (Hakan Bagci, 2016)



Gambar 2.1. Mekanisme RPC (*Remote Procedure Call*)

Proses berjalannya RPC di tunjukkan pada Gambar 2.1 yang di awali pada tahap 1, yaitu pengguna (*user*) mengakses aplikasi *client*. Kemudian dilanjutkan ke tahap 2, aplikasi *client* menghubungi RPC *server* melalui internet. Di tahap 3, RPC *server* menjalankan prosedur yang dibutuhkan oleh *client*. Kemudian hasil dari proses tersebut masuk ke tahap 4, yaitu mengirim hasil kepada aplikasi *client* melalui internet. Selanjutnya pada tahap 5, aplikasi *client* mengolah data yang diterima dan ditampilkan ke pengguna.

### 2.2.4. Framework Aplikasi

*Framework* pada dasarnya adalah alat bantu untuk membangun aplikasi sehingga terhindar dari *bug* dan menghemat waktu. *Framework* memiliki aturan dan arsitektur sehingga memungkinkan untuk membuat berbagai jenis aplikasi. (Dasari Hermitha Curie, 2019)

Aturan-aturan yang ada pada *framework* bergantung pada arsitekturnya. Misalkan, salah satu arsitektur adalah *Model-View-Controller* (MVC) yang memisahkan pemodelan data, alur bisnis, dan tampilan. *Framework* dengan arsitektur ini, memiliki aturan yang mengharuskan pemisahan ketiga bagian tersebut sehingga tidak dapat melakukan hal-hal di luar aturan yang ada.

### 2.2.5. Javascript

Javascript adalah bahasa pemrograman dinamis yang dapat menambahkan interaktivitas ke situs web. (Mozilla, 2020) Dengan Javascript, halaman web dapat memiliki animasi, tombol yang dapat diklik, hingga menampilkan *pop-up*.Web *browser* terkenal seperti Firefox dan Google Chrome menjadikan Javascript sebagai bahasa pemrograman standar bagi setiap halaman web yang dimuat.

Dengan Javascript, pengembang dapat mengelola halaman web melalui *Document Object Model* (DOM). DOM merupakan cara agar pengembang dapat memanipulasi elemen-elemen yang terdapat pada halaman web, seperti memunculkan teks, menghilangkan elemen tertentu, hingga melakukan hal tertentu saat sebuah tombol diklik oleh pengguna.

### 2.2.6. NodeJS

NodeJS adalah perangkat lunak untuk menjalankan program Javascript. Dengan menggunakan metode asingkron dalam manajemen prosesnya, NodeJS memungkinkan untuk membangun aplikasi jaringan yang dapat diskalakan. (NodeJS, 2020)

NodeJS biasa digunakan untuk membuat aplikasi berbasis Web. Dengan demikian memungkinkan Javascript berjalan di sisi *client* dan juga di sisi *server* sehingga dapat mengurangi beban pengembang dalam mempelajari maupun menerapkan bahasa pemrograman lainnya.

Javascript yang sebelumnya hanya berjalan di sisi *client*, kini memiliki cakupan hingga sisi *server* dengan memanfaatkan NodeJS. Maka, pengembang mendapat keuntungan berupa kode dari Javascript dapat mengakses hal-hal yang berkaitan dengan sistem operasi, seperti mengelola *file*, mengakses DBMS (*Database Management System*), serta mengelola protokol jaringan.

### 2.2.7. Javascript Object Notation (JSON)

JSON adalah format pertukaran data yang tidak bergantung pada bahasa pemrograman tertentu. Penulisan JSON berdasarkan aturan penulisan object dalam bahasa pemrograman Javascript. Data yang ditulis pada JSON, berbentuk name-value, artinya tiap isian data akan diberi nama. (Lindsay Bassett, 2015)

Pengiriman data yang dilakukan antara *client* dengan *server*, umumnya memerlukan sebuah format data yang dikenali oleh keduanya. Hal ini bertujuan agar dapat memproses data yang dikirim atau diterima. JSON adalah salah satu format data yang dapat diterapkan dalam kasus ini.

BAB III

# METODE PENELITIAN

## 3.1. Peralatan

Dalam melaksanakan penelitian digunakan peralatan untuk menunjang berjalannya pembuatan *framework*.

### 3.1.1. Perangkat Keras

Adapun perangkat keras yang dibutuhkan dalam proses membangun *framework* ini, yaitu komputer dengan:

1. *processor* Intel Core i5
2. *memory* 8GB
3. Solid State Disk (SSD) 120GB

### 3.1.2. Perangkat Lunak

Adapun perangkat lunak yang dibutuhkan dalam proses membangun *framework* ini, yaitu:

1. Arch Linux sebagai sistem operasi
2. Chromium sebagai *web browser*
3. VSCodium sebagai penyunting kode
4. NodeJS untuk menjalankan *framework*
5. Git untuk memanajemen versi

## 3.2. Perancangan Framework

### 3.2.1. Komponen Utama Framework

*Framework* yang dibuat dalam penelitian ini terdiri dari beberapa bagian (komponen), yaitu RPC Client, HTTP, dan RPC Server beserta prosedur-prosedur di dalamnya.



Keterangan warna:

: garis request

: garis response

Gambar 3.1. Bagian utama dari *framework*

Bagian utama yang membentuk *framework* ini ditunjukkan pada Gambar 3.1. Setiap bagian memiliki peran masing-masing dan saling terintegrasi bertujuan agar sistem dapat berjalan dengan baik.

RPC *server* berperan untuk melayani segala permintaan RPC *client* guna memenuhi kebutuhan pengguna. Di dalamnya terdapat berbagai prosedur yang dapat dijalankan saat dibutuhkan, terutama oleh RPC *client*.

HTTP (*HyperText Transfer Protocol*) merupakan protokol yang menjadi sarana RPC *client* berkomunikasi dengan RPC *server*. Protokol ini berdiri di atas jaringan komputer sehingga komunikasi dapat dilakukan dari manapun selama di dalam koneksi jaringan komputer.

RPC *client* berjalan pada antarmuka yang berperan sebagai media interaksi antara pengguna dengan aplikasi. Bila pengguna melakukan sebuah aksi pada antarmuka yang memerlukan RPC *server* maka RPC *client* akan menghubungi RPC *server* untuk melanjutkan pemrosesan.

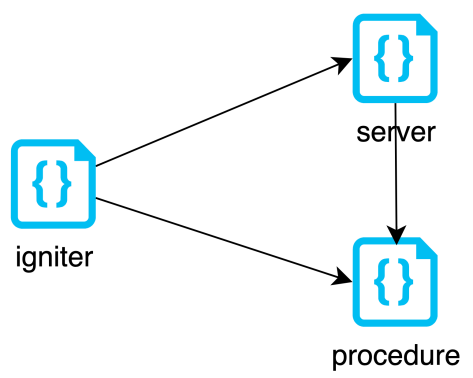
Prosedur merupakan aktivitas-aktivitas yang dapat dilakukan oleh RPC *server*. Tiap prosedur berisi kumpulan baris kode yang bertujuan untuk memenuhi kebutuhan tertentu secara spesifik. Prosedur juga dapat memiliki nilai balikan (*return value*)maupun tidak.

### 3.2.2. RPC *Server*

Dalam rancangan *framework* ini, RPC Server menyediakan layanan yang di dalamnya terdapat kumpulan prosedur. Tiap prosedur dibuat sesuai dengan kebutuhan masing-masing sistem yang dikembangkan. Kemudian prosedur-prosedur tersebut, digunakan oleh sisi *client* dengan cara *Remote Procedure Call*.

Saat sebuah prosedur digunakan oleh *client,* maka *server* akan menjalankannya dan mengirimkan kembali data yang dihasilkan. Jika prosedur tersebut tidak menghasilkan data, maka tidak ada data yang dikirimkan kembali ke *client*.

RPC *server* terdiri dari beberapa bagian, yaitu *igniter*, *server*, dan *procedure*. Tiap bagian memiliki tugasnya masing-masing dan saling mendukung agar dapat memenuhi kebutuhan sistem dengan baik.



Gambar 3.2. Bagian utama RPC *server*

Gambar 3.2 menunjukkan bagaimana tiap bagian pada RPC *server* saling bekerjasama. *Igniter* sebagai titik awal berperan menyalakan sistem. Sebelum sistem dapat dinyalakan, *igniter* akan mengatur *server* lebih dahulu.

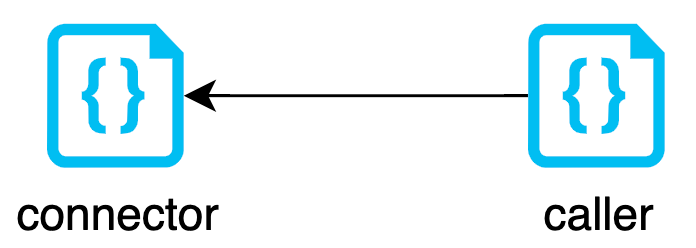
Pengaturan *server* yang dilakukan *igniter* memerlukan data-data dari *procedure*. Setelah *procedure* siap, *server* dapat dijalankan oleh *igniter*. Sampai di tahap ini, RPC *server* dari sistem sudah menyala.

### 3.2.3. RPC *Client*

Tugas utama dari RPC *client* adalah menyambungkan aplikasi dengan RPC *server*. Kemudian, aplikasi dapat memerintahkan server untuk menjalankan prosedur melalui RPC *client*.

Dalam perancangan ini, RPC *client* dapat mengelola data yang dihasilkan dari pemanggilan prosedur. Pengelolaan data dilakukan saat prosedur menghasilkan data maupun tidak. Jika prosedur tidak menghasilkan data, maka RPC *client* akan memberi informasi bahwa tidak ada data yang didapatkan.

RPC *client* terdiri dari 2 bagian, yaitu *connector* dan *caller*. Keduanya memiliki tugasnya masing-masing dan saling mendukung agar dapat memenuhi kebutuhan sistem dengan baik.



Gambar 3.3. Bagian utama RPC *client*

Gambar 3.3 menunjukkan bagaimana tiap bagian pada RPC *client* saling bekerjasama. *Caller* dapat dijalankan ketika pemanggilan prosedur dibutuhkan. Sebelum dapat dijalankan, *caller* akan menghubungkan dirinya dengan RPC *server* melalui *connector*.

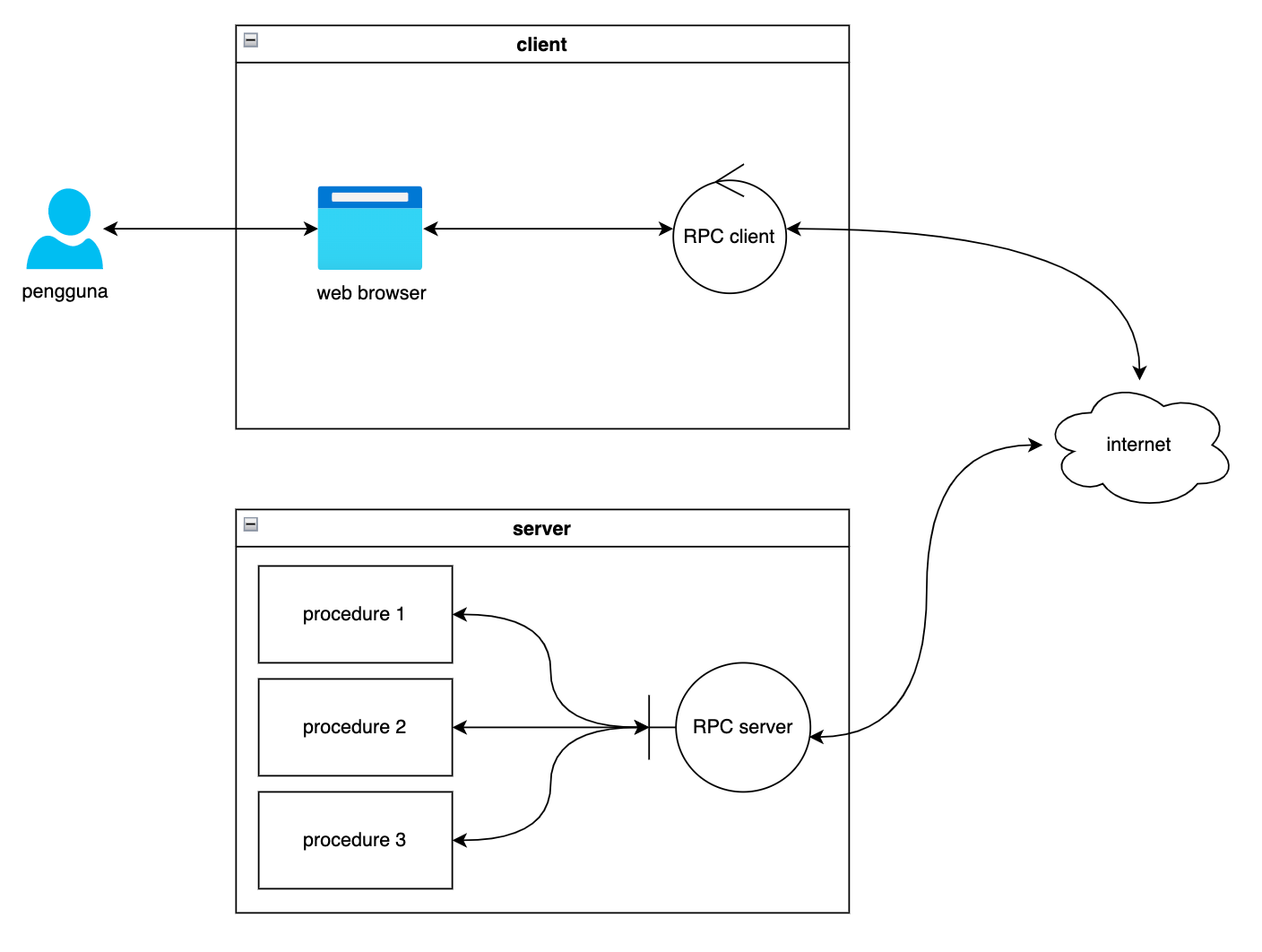
*Connector* hanya dijalankan sekali di awal, pada proses *caller* berikutnya, *connector* sebelumnya akan digunakan ulang. Dengan cara ini, diharapkan pemanggilan prosedur menjadi efisien, karena *caller* tidak perlu melakukan proses *connector* terus-menerus.

## 3.3. Perancangan Aplikasi

Perancangan aplikasi merupakan tahapan untuk menggambarkan bagaimana sebuah aplikasi dibagun dengan mengacu pada kebutuhan-kebutuhan fungsionalitas. Dalam perancangan ini terdiri atas diagram *usecase*, diagram *sequence*, diagram *activity*, rancangan tabel, dan rancangan antarmuka.

### 3.3.1. Arsitektur Sistem

Aplikasi Manajemen Karyawan dibangun dengan beberapa lapisan teknologi yang saling terintegrasi. Adapun bagian utamanya dapat dilihat pada diagram berikut.



Gambar 3.4. Diagram arsitektur sistem

Secara umum, arsitektur Aplikasi Manajemen Karyawan terbagi 2 bagian seperti pada Gambar 3.2, yaitu *client* dan *server*. *Client* berguna sebagai tempat interaksi antara aplikasi dengan pengguna. Pengguna dapat mengakses *client* dengan menggunakan *web browser*.

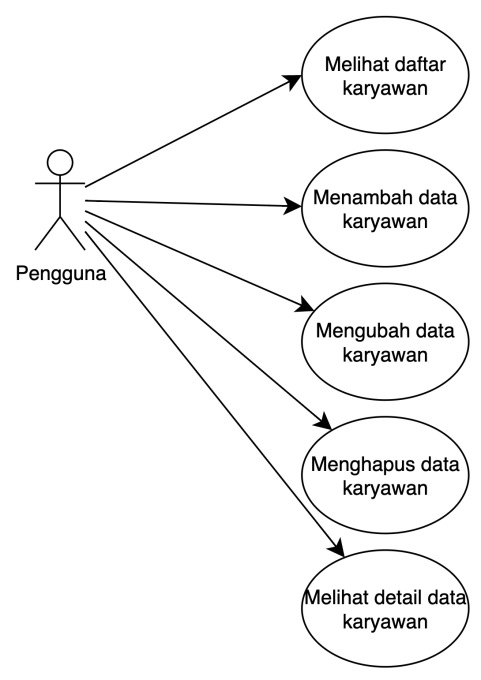
Pada bagian *client*, terdapat RPC *client* yang terkoneksi dengan *server* melalui Internet. Kemudian *client* dapat mengirim dan menerima data melalui koneksi yang aktif tersebut.

Di sisi *server*, pengiriman dan penerimaan data, serta koneksi *client* diterima dan dilayani oleh RPC *server*. Bagian inilah yang akan mengarahkan prosedur mana yang harus dijalankan saat interaksi dengan *client* terjadi.

3.3.2. RPC

### 3.3.2. Diagram Usecase

Diagram *usecase* adalah gambaran dari beberapa atau semua aktor, *usecase*, dan interaksi di antara keduanya. Berikut ini adalah diagram *usecase* aplikasi manajemen data karyawan.



Gambar 3.5. Diagram *usecase*

Pada Gambar 3.3, terdapat gambaran aktivitas pengguna terhadap aplikasi. Pengguna dapat memanajemen data karyawan, seperti melihat daftar karyawan, menambah data karyawan, mengubah data karyawan, menghapus data karyawan, dan melihat data karyawan secara detail.

### 3.3.3. Rancangan Tabel

Rancangan tabel menunjukkan bagaimana tabel digunakan untuk menyimpan data dalam pusat data. Pada aplikasi ini, terdapat 1 tabel yang menyimpan seluruh data. Berikut ini tabel tersebut.

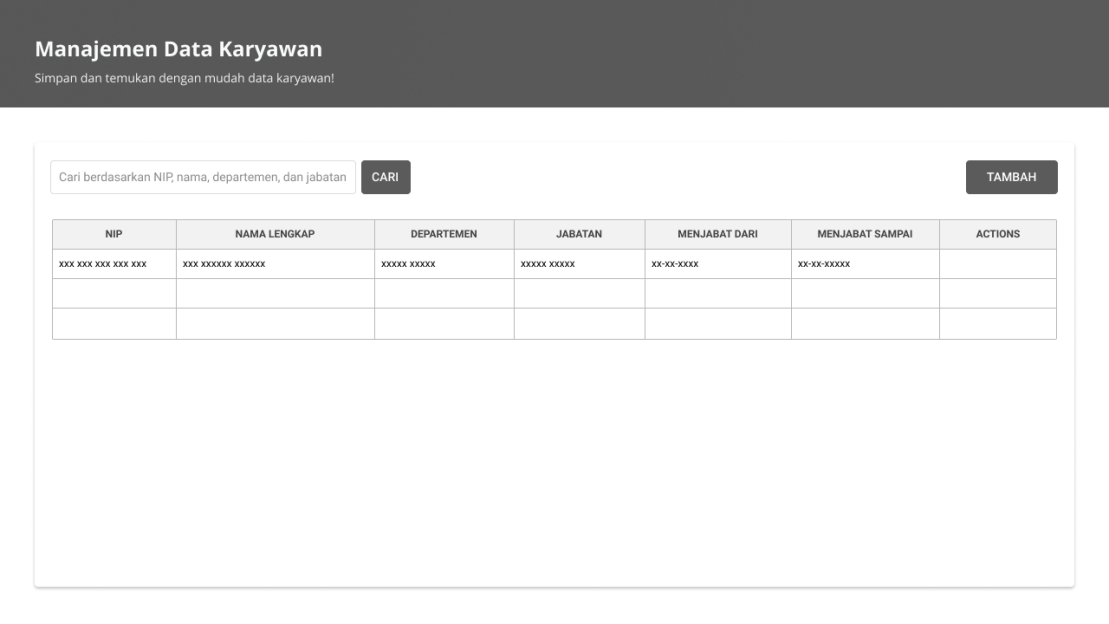
Tabel 3.13. Struktur tabel karyawan

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nomor** | **Nama Kolom** | **Tipe Data** | **Panjang Data** | **Keterangan** |
| 1 | id | Integer | 7 | identitas unik setiap data |
| 2 | nip | Varchar | 9 | kode unik kepegawaian |
| 3 | nama | Varchar | 35 | nama lengkap pegawai |
| 4 | departemen | Varchar | 15 | nama bagian tempat pegawai ditugaskan |
| 5 | jabatan | Varchar | 15 | nama tingkatan pegawai dalam instansi |
| 6 | menjabat\_dari | Date | 32 | tanggal mulai menjabat |
| 7 | menjabat\_sampai | Date | 32 | tanggal berakhirnya jabatan |

Pada tabel di atas, seluruh data karyawan di simpan pada penyimpanan karyawan yang digunakan untuk mengelola data-data tiap karyawan.

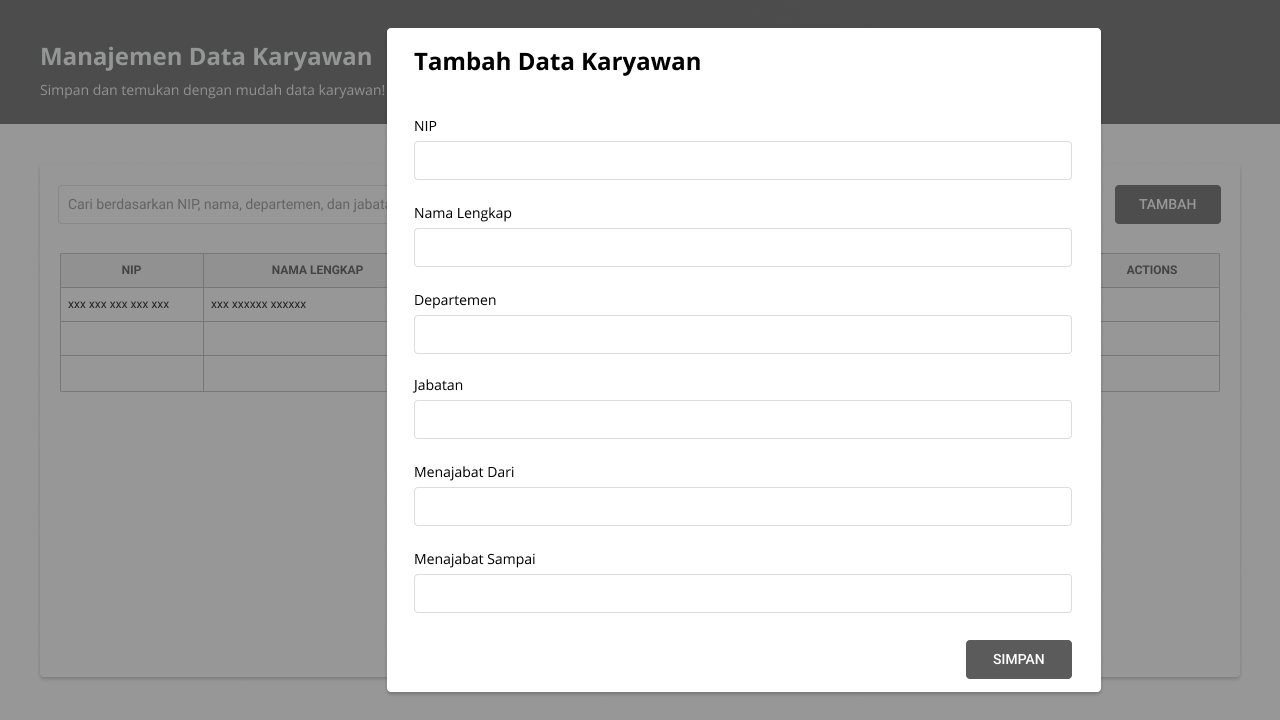
### 3.3.4. Rancangan Antarmuka

Rancangan antarmuka ini berupa gambaran umum desain tampilan yang akan digunakan saat proses pengembangan sistem. Setiap rancangan antarmuka dibuat sesuai dengan kebutuhan sistem yang tertuang dalam diagram usecase pada Gambar 3.5.



Gambar 3.6. Rancangan antarmuka daftar karyawan

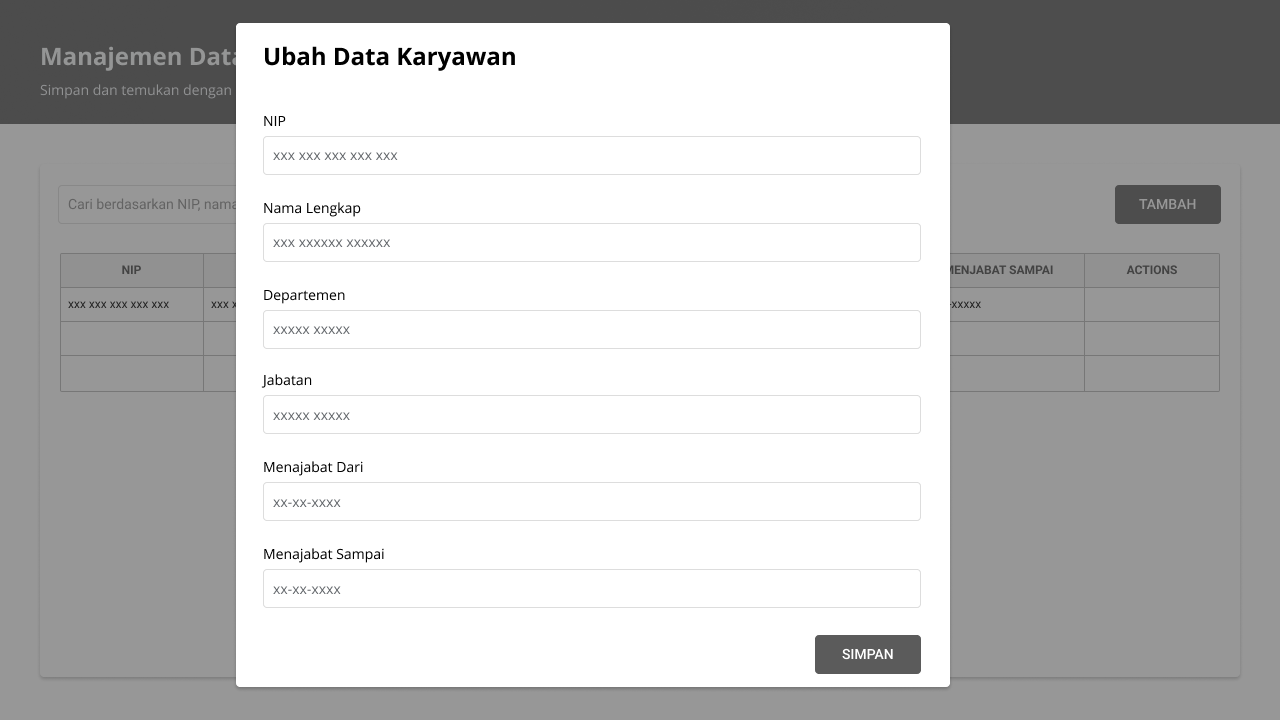
Gambar di atas merupakan halaman depan aplikasi saat pertama kali dibuka. Pada gambar tersebut, terdapat beberapa bagian utama, yaitu judul aplikasi yang bertuliskan “Manajemen Data Karyawan” dengan deskripsi “Simpan dan temukan dengan mudah data karyawan!” sebagai informasi pelengkap. Kemudian terdapat input pencarian, tombol untuk membuka halaman tambah data, dan tabel untuk menampilkan seluruh data karyawan yang tersedia.



Gambar 3.7. Rancangan antarmuka menambah data karyawan

Gambar di atas merupakan dialog yang berisi input untuk menambah data karyawan. Setelah seluruh input diisi, lalu tombol simpan ditekan, maka aplikasi akan menyimpan data tersebut.

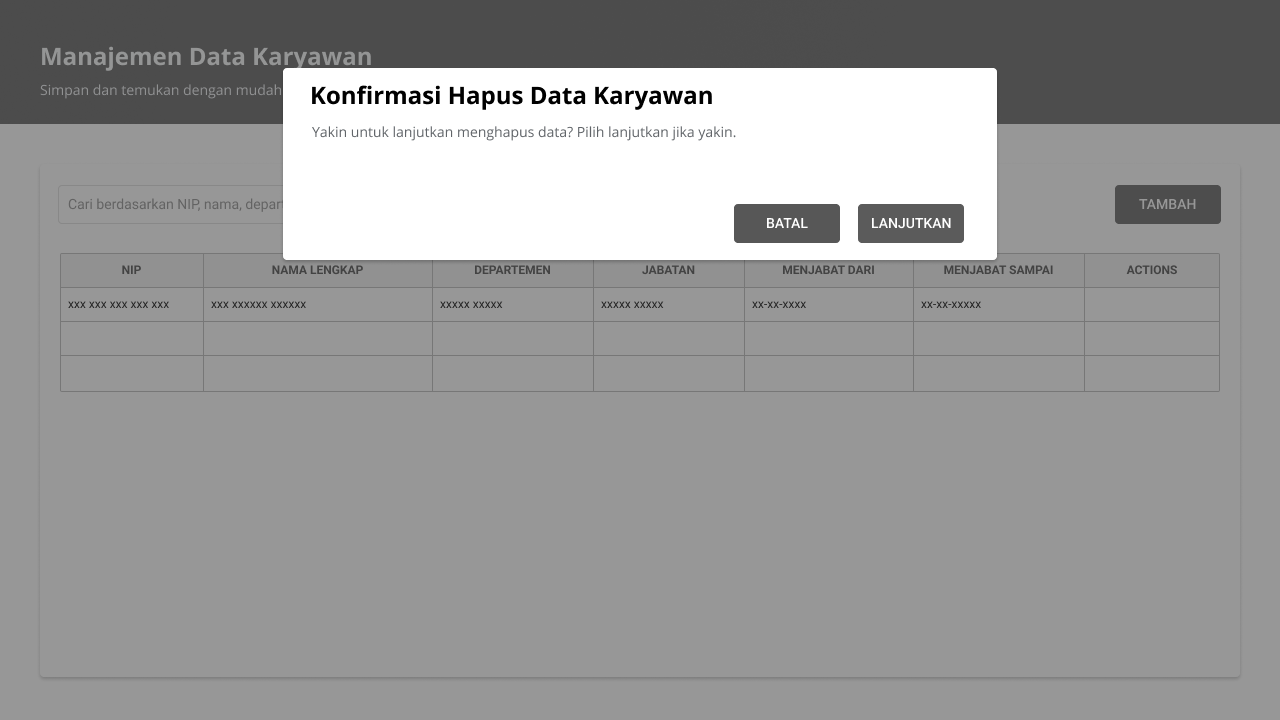
Data-data yang diperlukan untuk diinput, yaitu NIP, Nama Lengkap, Departemen, Jabatan, Menjabat Dari, dan Menjabat Sampai. Semuanya dibutuhkan untuk melengkapi data yang tertera pada Tabel 3.13.



Gambar 3.16. Rancangan antarmuka mengubah data karyawan

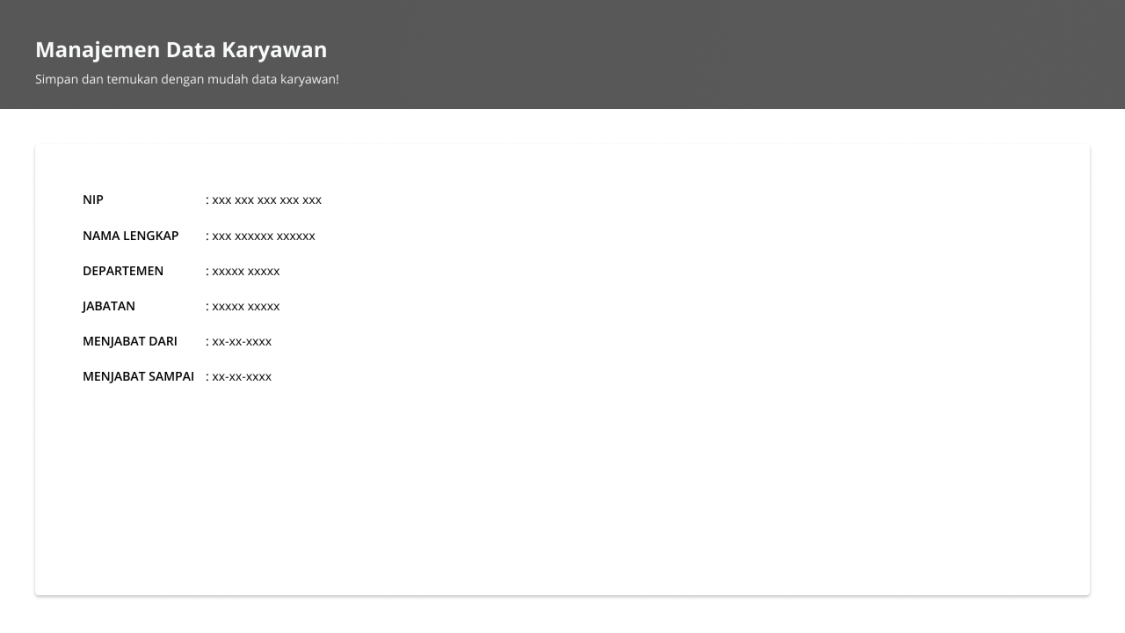
Gambar 3.16 merupakan halaman untuk mengubah data karyawan yang berisi data sebelumnya untuk diubah pada formulir input. Setelah input diisi, lalu tombol simpan ditekan, maka data karyawan akan diperbarui sesuai dengan yang perubahan yang dilakukan.

Data-data input sama seperti saat menambah data karyawan, perbedaannya adalah data input sudah diisi dengan data yang sebelumnya.



Gambar 3.17. Rancangan antarmuka konfirmasi hapus data karyawan

Gambar 3.17 merupakan tampilan dialog konfirmasi penghapusan data. Munculnya dialog ini, berguna untuk memastikan kembali dan mengurangi kesalahan saat pengguna akan menghapus data karyawan. Bila tombol lanjutkan ditekan, data karyawan terkait akan terhapus. Sebaliknya, bila tombol batal ditekan, dialog konfirmasi akan tertutup dan proses menghapus dibatalkan.



Gambar 3.18. Rancangan antarmuka melihat detail data karyawan

Gambar di atas merupakan tampilan detail data karyawan. Setiap data karyawan dapat dilihat secara detail melalui halaman di atas.

**BAB IV**

# IMPLEMENTASI DAN PEMBAHASAN

## 4.1. Implementasi *Framework*

Berdasarkan analisis dan perancangan *framework* yang telah dilakukan, maka dilanjutkan proses penulisan kode program dengan harapan program *framework* yang dihasilkan nantinya mampu memenuhi kebutuhan secara optimal. Selanjutnya pada bab ini akan dibahas proses pembuatan *framework* sesuai dengan hasil perancangan.

### 4.1.1. *Igniter* di RPC *Server*

*Igniter* terdiri dari 2 buah fungsi, yaitu register dan listen. Tiap fungsi memiliki tugasnya masing-masing dan saling mendukung agar dapat sistem dapat bekerja dengan baik.

|  |
| --- |
| function Igniter () {  let location;  this.register = (loc) => {  if (!location) {  location = loc;  }  const pDirPath = path.join(location, 'procedures');  let files = fs.readdirSync(pDirPath);  files.forEach((file) => {  const pFilesPath = path.join(location, 'procedures', file);  const procedures = require(proceduresPath);  Object.keys(procedures).forEach((name) => {  core.def(name, procedures[name]);  });  });  return this;  };    this.listen = async (callback) => {  try {  core.listen(8080, callback);  } catch (error) {  return error;  }  };  return this;  } |

Pada kode program di atas, fungsi dengan nama register berperan untuk mencari folder bernama procedures dan membaca seluruh *file* di dalamnya. Apabila folder procedures tidak ditemukan, maka akan menghasilkan galat. Dari tiap *file* yang ada di dalam folder procedures, akan didaftarkan ke dalam koleksi data prosedur.

Kemudian, dalam kode program di atas terdapat pula fungsi listen. Fungsi ini berperan untuk menjalankan *server* dengan membawa seluruh prosedur yang sudah terdaftar. Dengan jalannya *server*, artinya RPC *client* sudah dapat menjalankan prosedur sesuai dengan kebutuhan.

### 4.1.2. *Procedure* di RPC *Server*

*Procedure* terdiri dari beberapa fungsi, yaitu def, exists, dan call. Tiap fungsi memiliki tugasnya masing-masing dan saling mendukung agar dapat sistem dapat bekerja dengan baik.

|  |
| --- |
| const procedures = {};  const dependencies = {};  const def = (name, handler) => {  procedures[name] = handler;  }; |

Pada kode di atas, fungsi def berguna untuk mendaftarkan prosedur berdasarkan nama dari prosedur tersebut. Proses pendaftaran prosedur dilakukan dengan membuat sebuah wadah untuk koleksi data prosedur. Kemudian memasukkan prosedur ke dalam wadah tersebut.

|  |
| --- |
| const exists = (name) => {  return !!procedures[name]  && typeof procedures[name] === 'function';  }; |

Pada kode di atas, fungsi exists mencari prosedur dengan nama tertentu. Jika ditemukan, maka akan menghasilkan true. Sebaliknya, jika tidak, maka akan menghasilkan false.

|  |
| --- |
| const call = async (name, parameters) => {  const fn = procedures[name];  const params = { dependencies };  parameters.forEach((p, i) => {  params[i] = p;  });  return await fn(...Object.values(params));  }; |

Pada kode di atas, fungsi call mencari prosedur berdasarkan nama tertentu. Kemudian mengatur parameter agar dapat dikirimkan saat prosedur dipanggil. Setelah seluruh parameter siap, prosedur dijalankan.

### 4.1.3. *Server* di RPC *Server*

*Server* terdiri dari 2 buah fungsi, yaitu matchProcedure dan listen. Tiap fungsi memiliki tugasnya masing-masing dan saling mendukung agar dapat sistem dapat bekerja dengan baik.

|  |
| --- |
| const procedure = require('./procedure');  const matchProcedure = async (req, res) => {  const { call, parameters } = JSON.parse(req.body);    if (procedure.exists(call)) {  let result = await procedure.call(call, parameters);  res.json(JSON.stringify(result));  } else {  throw Error(`Procedure "${data.call}" is undefined.`);  }  }; |

Pada kode di atas, fungsi matchProcedure menerima nama prosedur beserta parameternya. Kemudian mencari prosedur dengan nama tersebut melalui fungsi exists.

Jika prosedur ditemukan, maka prosedur tersebut dijalankan dengan membawa parameter yang sudah diterima sebelumnya. Kemudian hasilnya dikirimkan ke *client* dalam bentuk JSON.

Tetapi jika prosedur tidak ditemukan, maka akan mengirimkan *error* dengan pesan prosedur tidak ditemukan.

|  |
| --- |
| const server = new Server();  let rootURL = '/';  const listen = (port, callback) => {  server.post(rootURL, matchProcedure);  server.listen(port, callback);  }; |

Pada kode di atas, fungsi listen mendaftarkan fungsi matchProcedure ke dalam pemrosesan *server.* Setelah *server* siap dengan pemrosesan di dalamnya, selajutnya *server* di jalankan pada *port* tertentu.

### 4.1.4. *Connector* di RPC *Client*

*Connector* dituliskan dalam bentuk fungsi yang bernama connect. Fungsi ini memiliki tugasnya sendiri dalam keseluruhan sistem, yaitu sebagaii *connector* untuk sisi *client*.

|  |
| --- |
| self.connect = function connect(host) {  self.host = host;  return self;  }; |

Pada kode di atas, fungsi connect mengatur alamat dari *server*. Fungsi ini harus dijalankan di awal oleh *client* agar dapat terhubung ke *server* dan proses berikutnya dapat berjalan dengan baik.

### 4.1.5. *Caller* di RPC *Client*

*Caller* dituliskan dalam bentuk fungsi yang bernama call. Fungsi ini memiliki tugasnya sendiri dalam keseluruhan sistem.

|  |
| --- |
| self.call = function call(procedure, params) {  return new Promise(function(resolve, reject) {  const body = JSON.stringify({  call: procedure,  parameters: params,  });  const xhr = new XMLHttpRequest();  xhr.onreadystatechange = function onreadystatechange(e) {  if (e.target.readyState === 4) {  if (e.target.status == 200) {  try {  const parsedResult = JSON.parse(  e.target.responseText  );  resolve(parsedResult);  } catch(err) {  resolve(e.target.responseText);  }  }  reject(e);  }  };  xhr.open('POST', self.host);  xhr.send(body);  });  }; |

Pada kode di atas, fungsi call menyiapkan pengiriman data ke *server*. Kemudian setelah siap, pengiriman dilakukan melalui HTTP *request* dengan membawa data dalam format JSON.

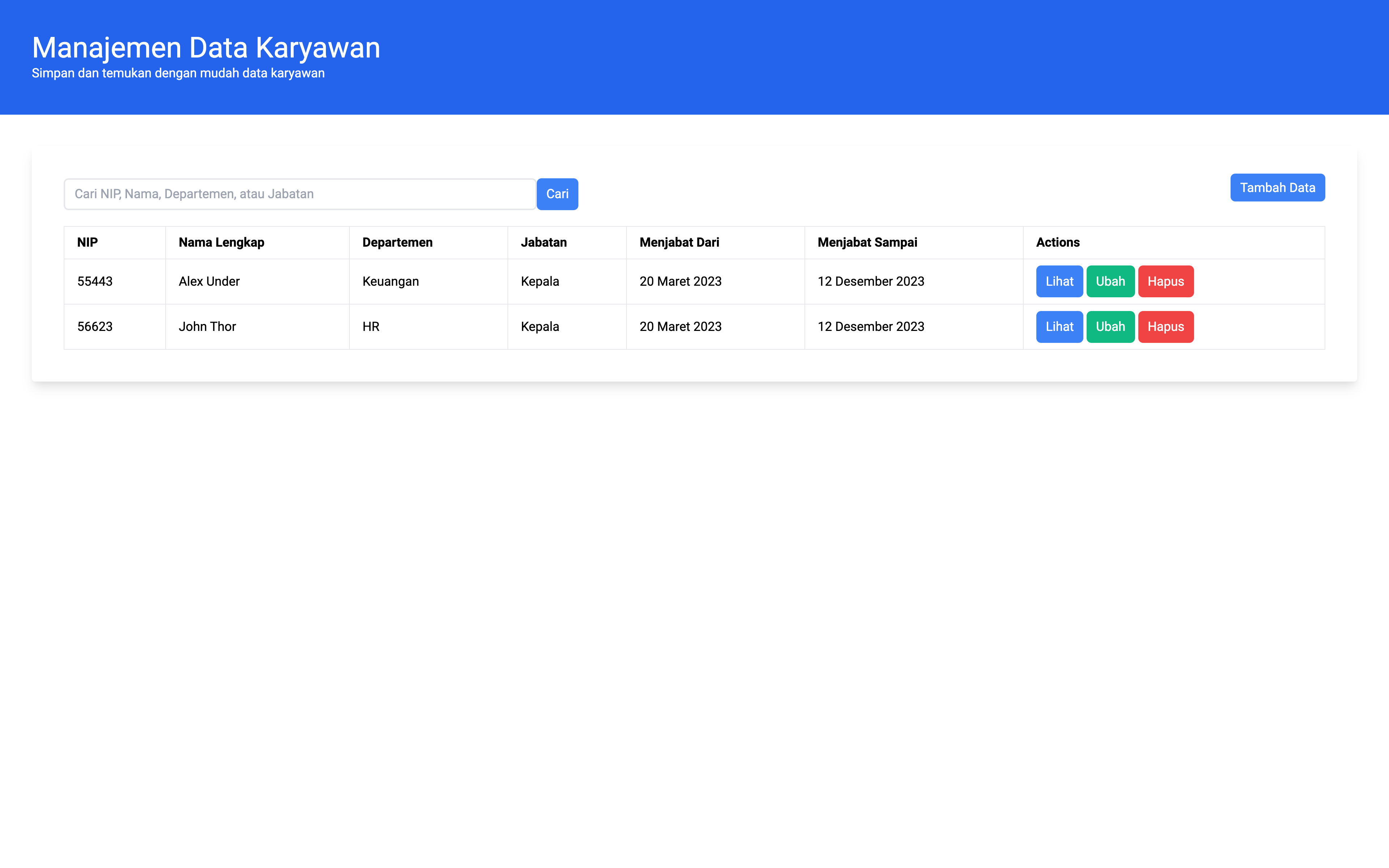
## 4.2. Implementasi Sistem

Berdasarkan analisis dan perancangan sistem yang telah dilakukan, maka dilanjutkan proses penulisan kode program dengan harapan program yang dihasilkan nantinya mampu memenuhi kebutuhan secara optimal. Selanjutnya pada bab ini akan dibahas proses pembuatan program serta tampilan-tampilan sesuai dengan hasil perancangan.

## 4.3. Tampilan Sistem

Sistem yang dibuat guna mengimplementasikan *framework* dalam penelitian ini disertai dengan tampilan-tampilan untuk memenuhi proses yang diperlukan. Tampilan-tampilan tersebut dibuat berdasarkan rancangan antarmuka yang telah dilakukan sebelumnya. Berikut ini adalah pembahasan dari tampilan-tampilan tersebut.

### 4.3.1. Halaman Depan



Gambar 2.3. Tampilan halaman depan

Pada Gambar 2.3, ditampilkan halaman depan yang merupakan halaman yang pertama kali terbuka saat sistem diakses. Halaman ini menampilkan seluruh data yang tersimpan pada sistem dalam bentuk tabel.

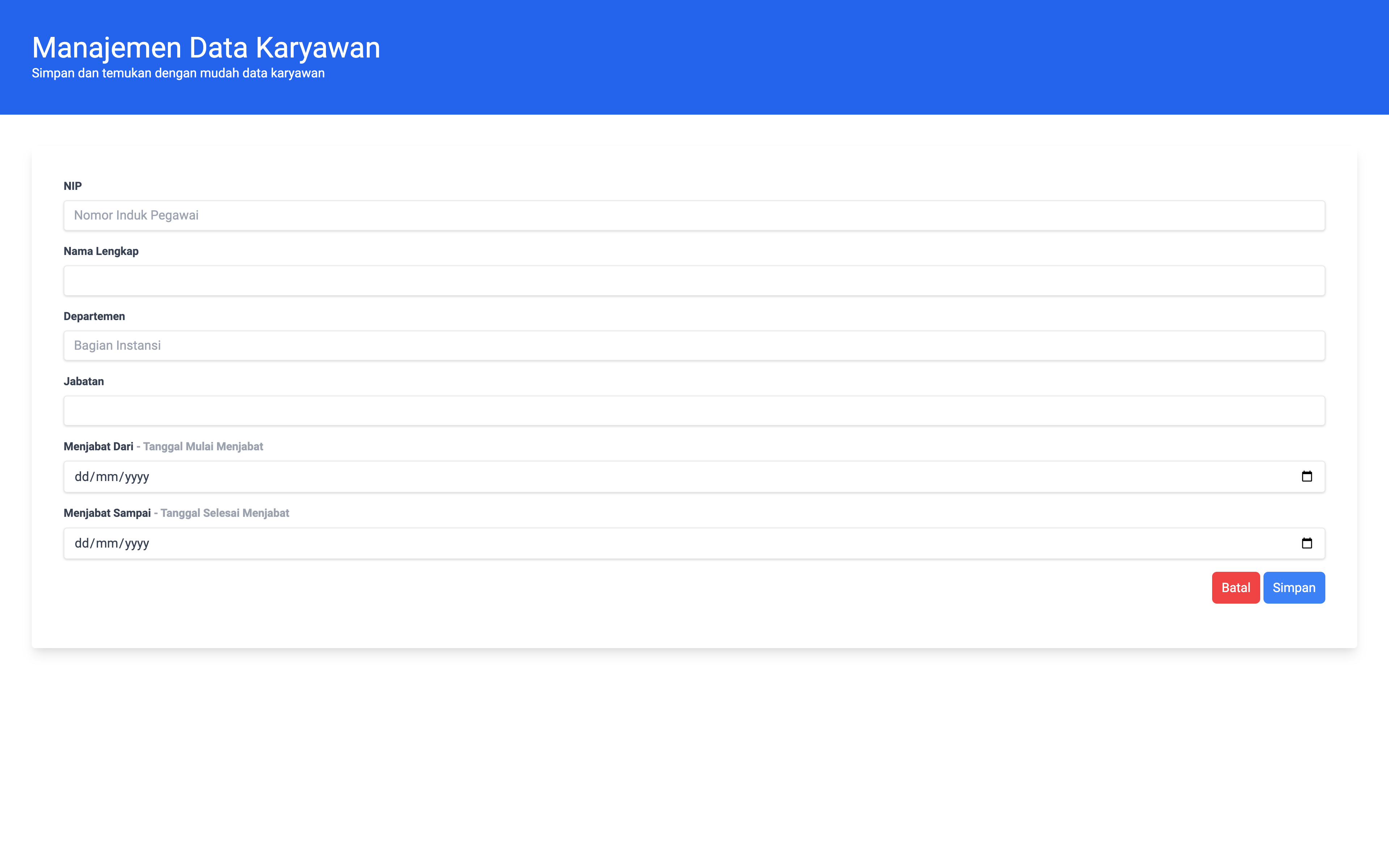
Tampilan tabel terdiri dari beberapa kolom, yaitu NIP, Nama Lengkap, Departemen, Jabatan, Menjabat Dari, Menjabat Sampai, dan *Actions*. Selain kolom *Actions*, kolom-kolom yang ada merupakan struktur dari data karyawan yang dikelola dalam sistem.

Kolom *Actions* digunakan untuk menampilkan tombol-tombol yang berguna untuk mengontrol setiap baris data. Kontrol yang dapat dilakukan, seperti lihat detail data, ubah data, dan hapus data.

Pada halaman ini juga tersedia input pencarian untuk mendukung kemampuan dalam menampilkan data. Data yang ditampilkan hanya yang sesuai dengan apa yang dicari melalui pencarian tersebut.

Selain itu, terdapat tombol tambah data yang akan mengarahkan ke halaman untuk menambah data. Seluruh data yang dimasukkan ke dalam sistem, melalui menekan tombol ini.

### 4.3.2. Halaman Tambah Data



Gambar 2.3. Tampilan halaman tambah data

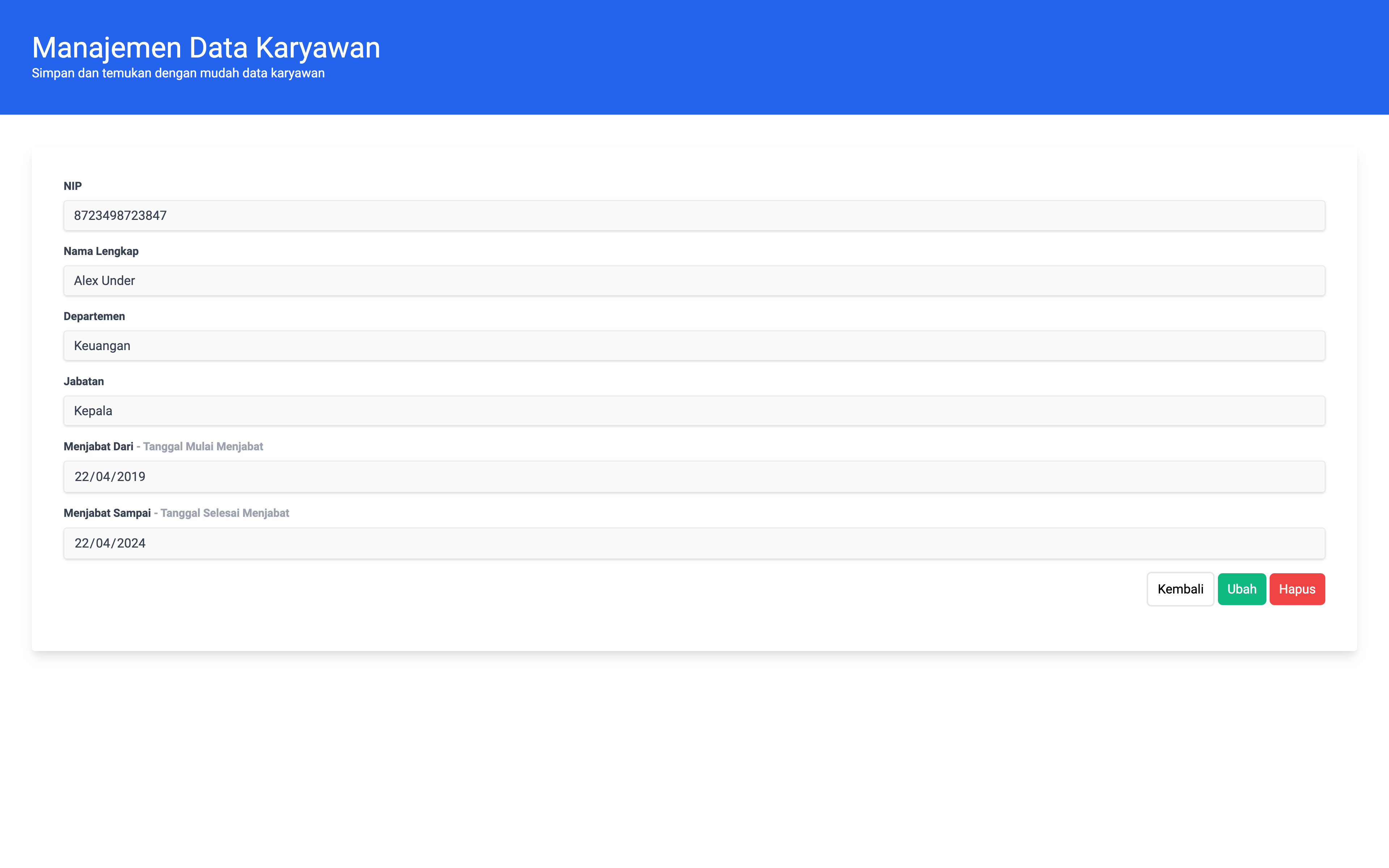
Pada Gambar 2.3, ditampilkan halaman untuk memasukkan data untuk disimpan ke dalam sistem. Dengan tersimpannya data baru tersebut, maka terjadilah proses penambahan data.

Data yang dimasukkan pada proses ini adalah NIP, Nama Lengkap, Depertemen, Jabatan, Menjabat Dari, dan Menjabat Sampai. Data-data tersebut adalah struktur dari data karyawan yang dikelola sistem.

Pada halaman ini terdapat tombol simpan yang berguna untuk mengirim data masukan ke dalam sistem. Setelah tombol ini bekerja, proses menyimpan data selesai.

Selain itu, terdapat pula tombol batal yang digunakan jika akan mengurungkan proses menambah data. Tombol ini akan mengabaikan data yang sudah dimasukkan dan kembali ke halaman depan.

### 4.3.3. Halaman Ubah Data



Gambar 2.3. Tampilan halaman ubah data

Pada Gambar 2.3, ditampilkan halaman untuk mengubah data. Proses yang terjadi di halaman ini adalah memasukkan keseluruhan atau sebagian data untuk disimpan sebagai pengganti dari data sebelumnya.

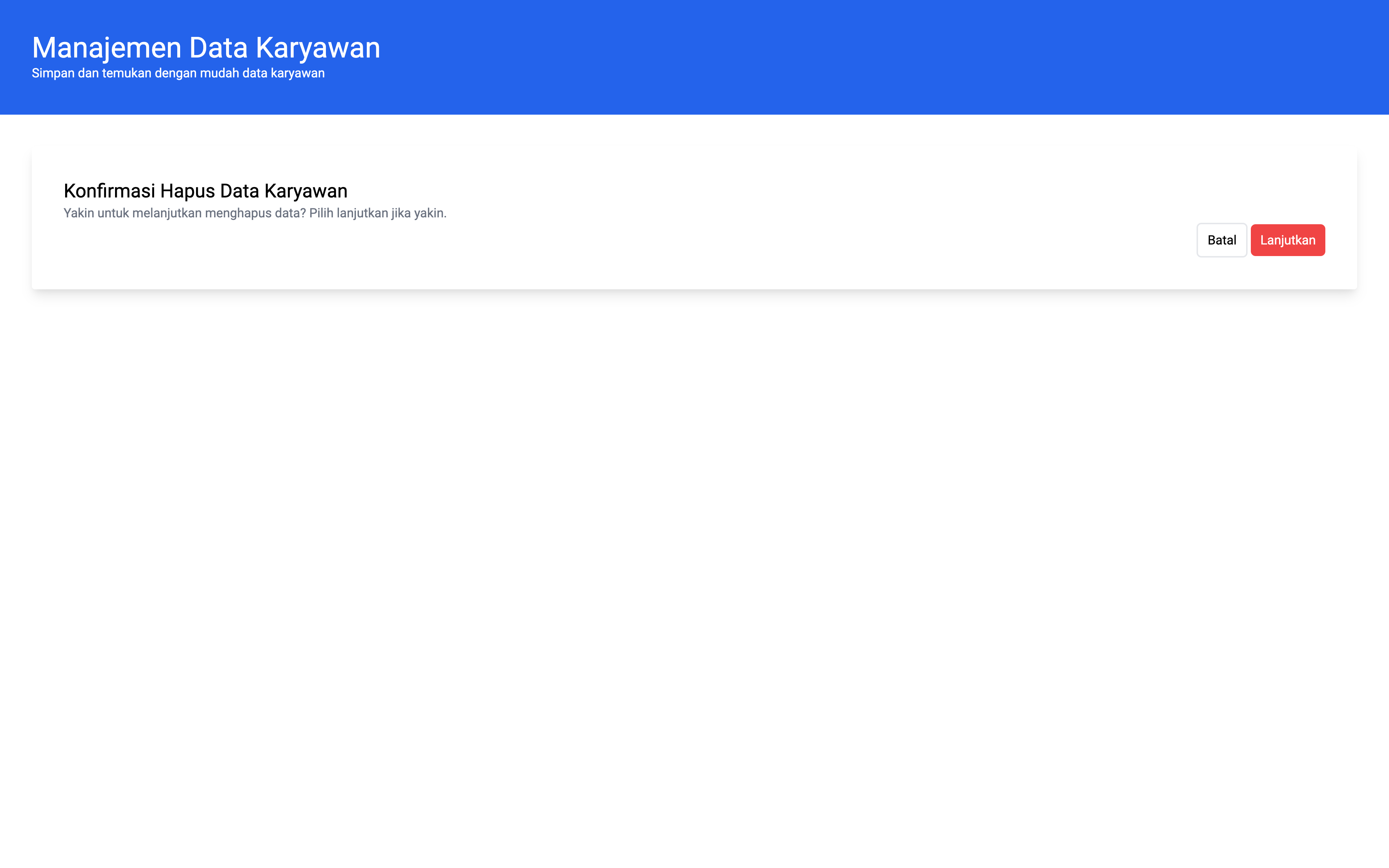
Agar proses mengubah data menjadi lebih mudah, mulanya halaman ini menampilkan data yang sudah tersimpan sebelumnya. Seluruh data yang pernah dimasukkan dapat dilakukan perubahan.

Pada halaman ini, terdapat tombol kembali yang digunakan untuk mengurungkan proses ubah data. Ketika tombol ini ditekan, maka akan kembali ke halaman depan.

Terdapat pula tombol ubah yang digunakan untuk menyimpan data yang telah diubah. Ketika tombol ini ditekan, data yang telah diubah akan dikirimkan ke dalam sistem dan disimpan.

### Selain itu, terdapat tombol hapus yang digunakan untuk menghapus data. Tombol ini berguna sebagai langkah singkat jika ingin menghapus data tanpa kembali ke halaman depan terlebih dahulu.

### 4.3.4. Halaman Hapus Data



Gambar 2.3. Tampilan halaman konfirmasi hapus data

Pada Gambar 2.3, ditampilkan halaman untuk menghapus data. Proses menghapus data dilakukan melalui sebuah dialog konfirmasi.

Dialog konfirmasi berguna mencegah kesalahan dalam menghapus data. Pencegahan seperti ini diperlukan karena menghapus data akan menghilangkan data tersebut dari sistem.

Pada halaman ini terdapat tombol batal yang digunakan jika ingin mengurungkan proses menghapus data. Tombol ini akan mengarahkan kembali ke halaman depan.

Selain itu, terdapat tombol lanjutkan yang berguna untuk melanjutkan proses menghapus data. Ketika tombol ini ditekan, sistem akan menghapus data tersebut sehingga tidak lagi tampil di tabel karyawan di halaman depan.

**BAB V**

PENUTUP

## 5.1. Simpulan

adsasd kjasdkljasjdklsajdj

asdasdasdasd

## 5.2. Saran

Adapun saran dari penelitian ini guna memperbaiki atau menyempurnakan pengembangan selanjutnya adalah meninjau dan memperbaiki beberapa fitur terutama untuk memenuhi kebutuhan otentikasi pengguna.

# DAFTAR PUSTAKA

Bagci, Hakan. 2016. *A Lightweight and High Performance Remote Procedure Call Framework for Cross Platform Communication*. Turkey: SCITEPRESS.

Curie, Dasari Hermitha. 2019. *Analysis on Web Frameworks*. India: Karunya Institute of Technology and Sciences.

Fredrich, Todd. 2019. *Learn REST: A RESTful Tutorial*.

https://restapitutorial.com. Diakses 10 Maret 2020.

Goralski, Walter. 2017. *The Illustrated Network*. United States: Elsevier Inc.

Mozilla. 2020. *Resources for developers, by developers*. https://developer.mozilla.org. Diakses 20 Agustus 2020.

Negara, Yosafat Aria. 2018. *Aplikasi Komunikasi Socket Client-Server Layanan*

*Kamar Hotel Berbasis Android Pada Jaringan Lokal Menggunakan TCP*

*IP*. Diploma thesis, STMIK Akakom Yogyakarta.

NodeJS. 2020. *NodeJS*. https://nodejs.org. Diakses 20 Agustus 2020.

Putranto, Bambang Purnomosidi Dwi. 2013. *Pengembangan Aplikasi Cloud*

*Menggunakan Node.js*. https://github.com/bpdp/buku-cloud-nodejs.

Diakses 10 Maret 2020.

Riko. 2019. *Implementasi REST API untuk Sistem Penjadwalan Pendadaran dan*

*Seminar Proposal Skripsi Mahasiswa (Studi Kasus Program Studi Sistem*

*Informasi STMIK Akakom Yogyakarta)*. Sistem Informasi STMIK

Akakom Yogyakarta

Satraji, Lalu Himawan. 2017. *Implementasi Web Service untuk Delivery Makanan Khas Yogyakarta Berbasis Mobile*. Teknik Informatika STMIK Akakom Yogyakarta.

Sejwal, Shobhika. 2019. *An Approach to Resolve Heterogeneity Using RPC in Client Server Systems*. India: Amity University.

Srinivasan, R. 1995. *RPC: Remote Procedure Call Protocol Specification Version*

*2.* Sun Microsystems.

Sulistyo, Iqbal. 2017. *Implementasi Teknologi Responsive Web pada Sistem Informasi Tilang Berbasis Codeigniter*. Teknik Informatika STMIK Akakom Yogyakarta.

Wawan, Setiawan. 2017. *Absensi Siswa dengan Teknologi NodeJS Studi Kasus SMKN 1 Sawit*. Teknik Informatika Sekolah Tinggi Manajemen Informatika dan Komputer Akakom.